Android测试

# Android测试手段

（1）交由第三方云测平台进行

（2）利用Android自动化测试框架自己编写

# 第三方云测平台

例如：

[（1）腾讯优测云测试平台](http://utest.21kunpeng.com/)

[（2）Testin](https://www.testin.cn)

[（3）阿里云测试平台](http://mqc.aliyun.com)

**优点：**丰富的测试能力，提供兼容性测试、功能测试、性能测试、安

全测试、报告生成等

**缺点：**付费

# Android自动化测试框架

## 本地单元测试

主要运行没有依赖于Android框架或可以模拟Android 框架依赖项时的单元测试

### JUnit 4 framework

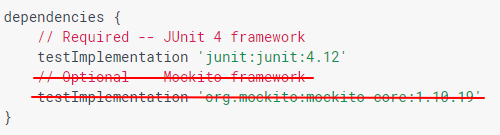
#### 使用场景

1，适用于依赖比较简单的类方法验证

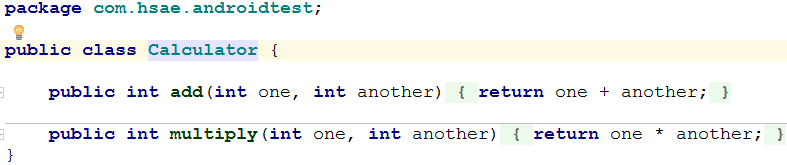
2，适用于有返回值的方法验证，比如工具类方法等

#### 简单例子

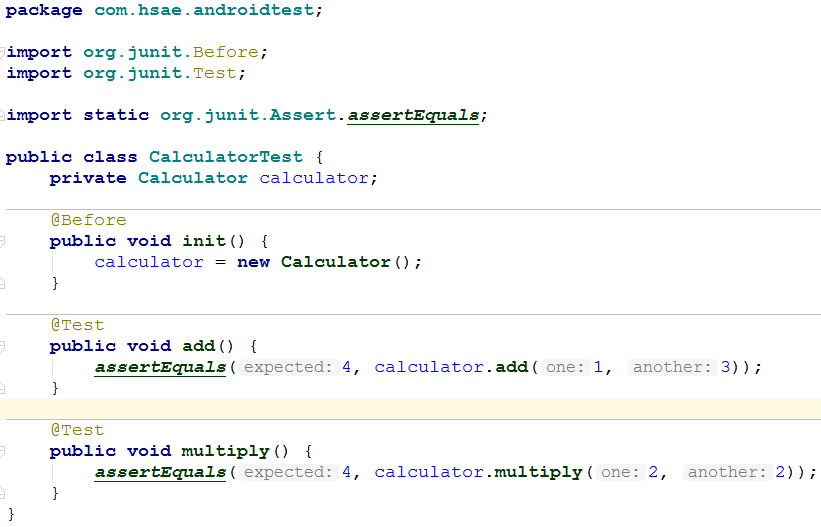
##### 添加依赖



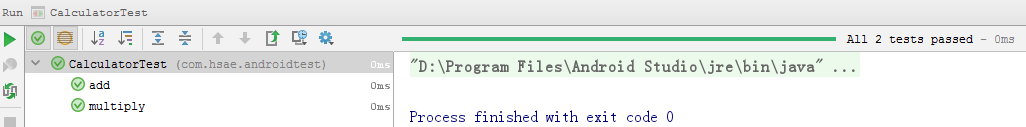
##### 待测试类



##### 测试类（需在module-name/src/test/java目录下）



##### 执行结果



#### JUnit 4简单说明

##### 常用注解

@Test：表明是一个测试方法

@Before：在每个测试方法之前调用

@After：在每个测试方法之后调用

@BeforeClass：在类中所有方法前运行，必须是static void方法

@AfterClass：在类中所有方法后运行，必须是static void方法

##### 执行顺序

@BeforeClass->@Before->@Test->@After->@AfterClass

##### 优缺点

优点：执行速度快，一般用于工具类

缺点：只能测试有返回值的方法，对复杂依赖无法测试

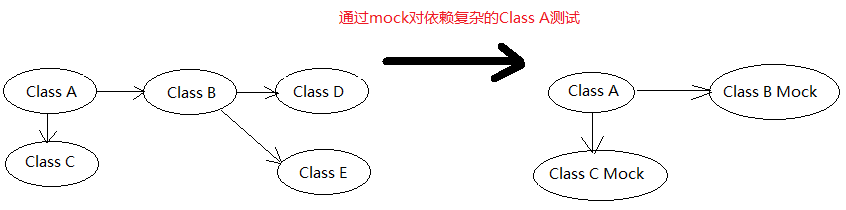
#### 更深入挖掘请参考官网

<https://junit.org/junit4/>

### Mockito framework

#### 使用场景

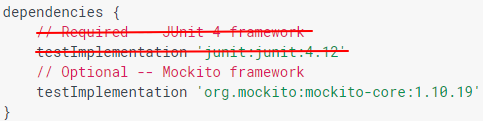
1，可以对依赖复杂的类方法进行验证（模拟对象行为）



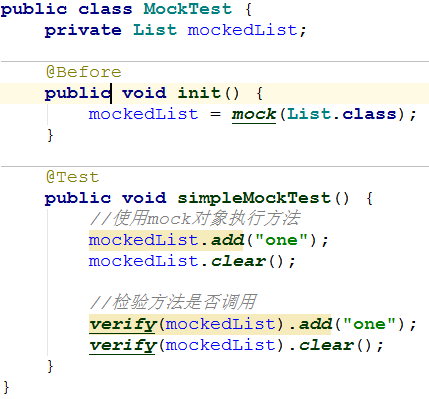
2，可以对无返回值的方法进行验证（检验一个方法调用情况）

#### 简单例子

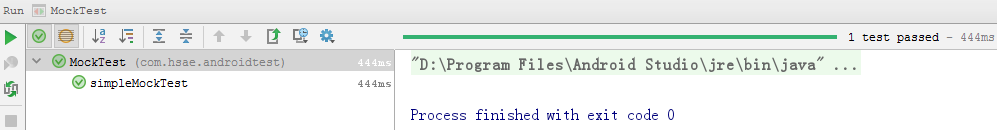
##### 添加依赖



##### 测试类（需在module-name/src/test/java目录下）



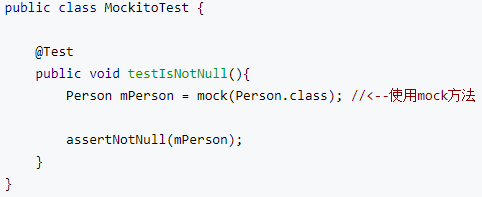
##### 执行结果



#### Mockito简单说明

##### Mock的四种方式

###### 使用mock方法



###### 注解方法



###### 运行器方法



###### MockitoRule方法



##### 缺陷

不能mock静态、final、私有方法等

##### 更深入挖掘请参考文档

<https://github.com/hehonghui/mockito-doc-zh>

## 仪器单元测试（依赖Android框架）

在Android设备或模拟器上运行的单元测试

### Espresso

#### 使用场景

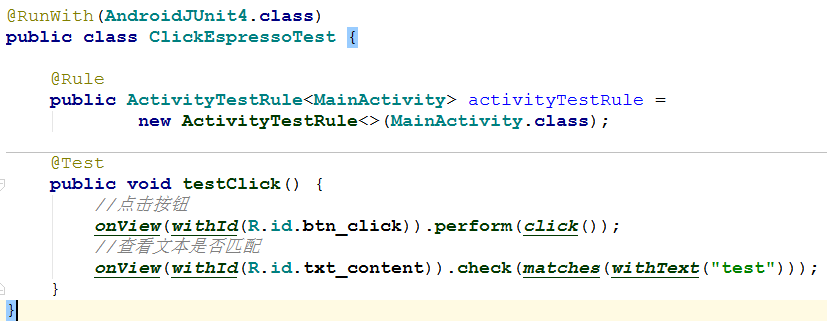
只能在本应用程序中测试UI，不支持跨进程交互

#### 简单例子

##### 添加依赖项&设备运行器



##### 新增一个测试类

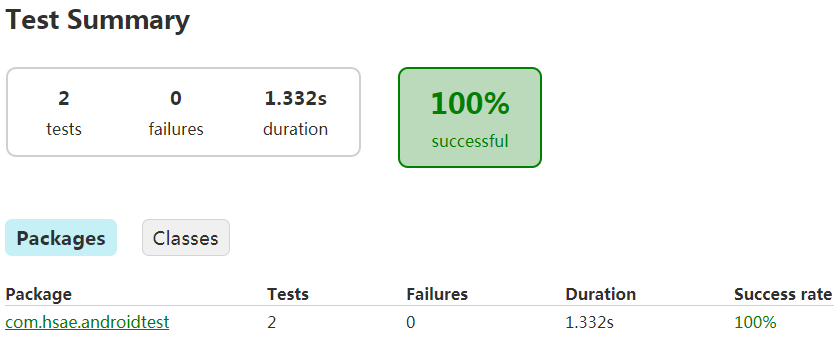


##### 执行结果

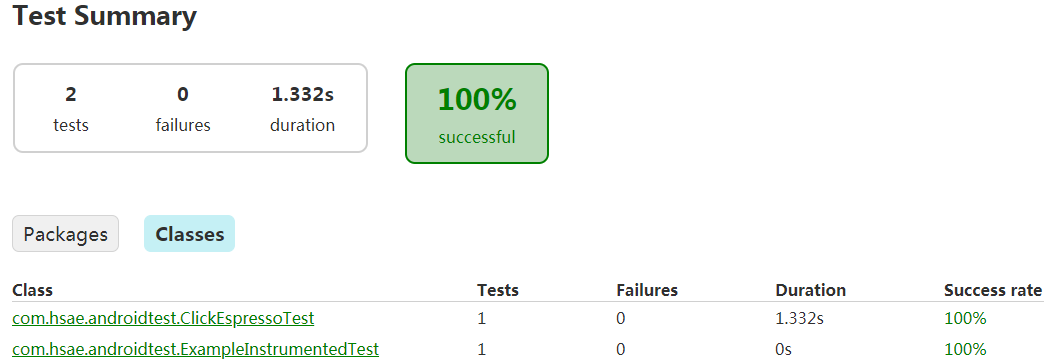
执行gradlew connectedAndroidTest命令，进入

$build/reports/androidTests/connected/目录下，打开index.html，可查看执行结果。

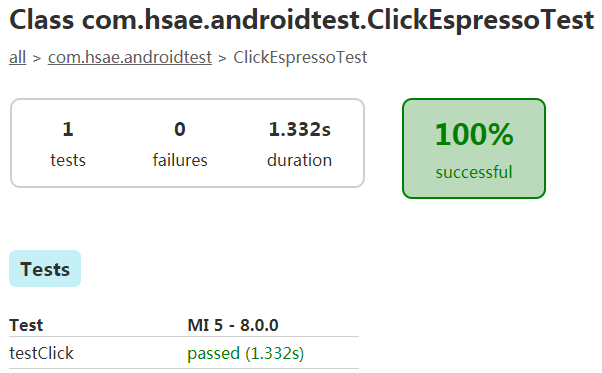
**总体测试概要：**



**所有类概要：**



**具体某个方法测试情形：**



#### 简单说明

##### 常用的功能有哪些

###### 查找

通过onView()方法找寻指定元素，可以根据控件ID或文本

###### 操作

通过perform()方法可以执行一些操作，比如点击(click())等

###### 检查

通过check()方法检查操作是否符合我们的预期

##### 注意点

需要关闭测试设备上的动画，在测试设备上打开系统动画可能导致意外结果或者导致测试失败。具体在开发者选项中关闭以下所有项：

1）窗口动画缩放

2）过渡动画缩放

3）动画程序时长缩放

##### 优点

1）提供了测试操作与正在测试的应用程序界面自动同步

2）在主线程空闲状态时运行测试命令，提高了测试可靠性

#### 更深入挖掘请参考文档

<https://developer.android.google.cn/training/testing/espresso/>

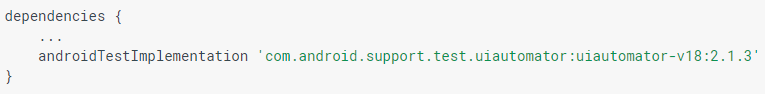
### UI Automator

#### 使用场景

可以在多个应用程序中测试UI，跨进程测试

#### 简单例子

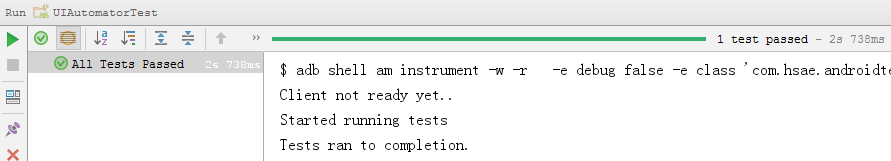
##### 添加依赖



##### 创建一个测试类



##### 执行结果



#### 简单说明

##### 常用API

###### UiDevice

设备对象，通过**UiDevice**.***getInstance***(instrumentation)

来获取。其可以检测设备的各种属性及执行设备级的操作，如：点击Home键、Back键等

###### UiSelector

用于获取某些符合条件的UI控件对象，可以通过资源ID及文本描述等获取

###### UiObjector

代表一个UI控件，通过它可以进行一些操作，如：单击、长按等

###### UiCollection

表示UI控件集合，相当于ViewGroup

###### UiScrollable

表示可以滚动的控件

##### 注意点

###### 支持最低版本

UI Automator库支持的最低Android版本为18，所以需要minSdkVersion大于等于18

###### 测试方法说明

测试方法需要以test开头，如果存在多个测试方法，以test后的字母顺序执行

##### 优缺点

优点：支持跨进程测试，Rom层级的测试

缺点：仅支持Native应用，对含H5网页应用支持不够好

#### 更深入挖掘请参考文档

<https://developer.android.google.cn/training/testing/ui-testing/uiautomator-testing>

## 测试的覆盖程度

用于检测是不是所有的代码都被测试过了？通过加入代码测试覆盖率的统计来进行查看控制

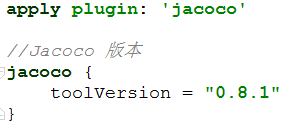
### JaCoCo

#### 原理

通过ASM框架进行注入修改字节码文件，在测试前对所有方法注入插桩，测试时，所有运行过的方法会生成动态覆盖信息到文件，最后统一对覆盖信息进行处理，生成报告

#### 一个例子

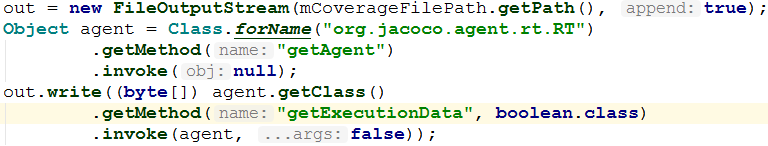
##### 添加依赖



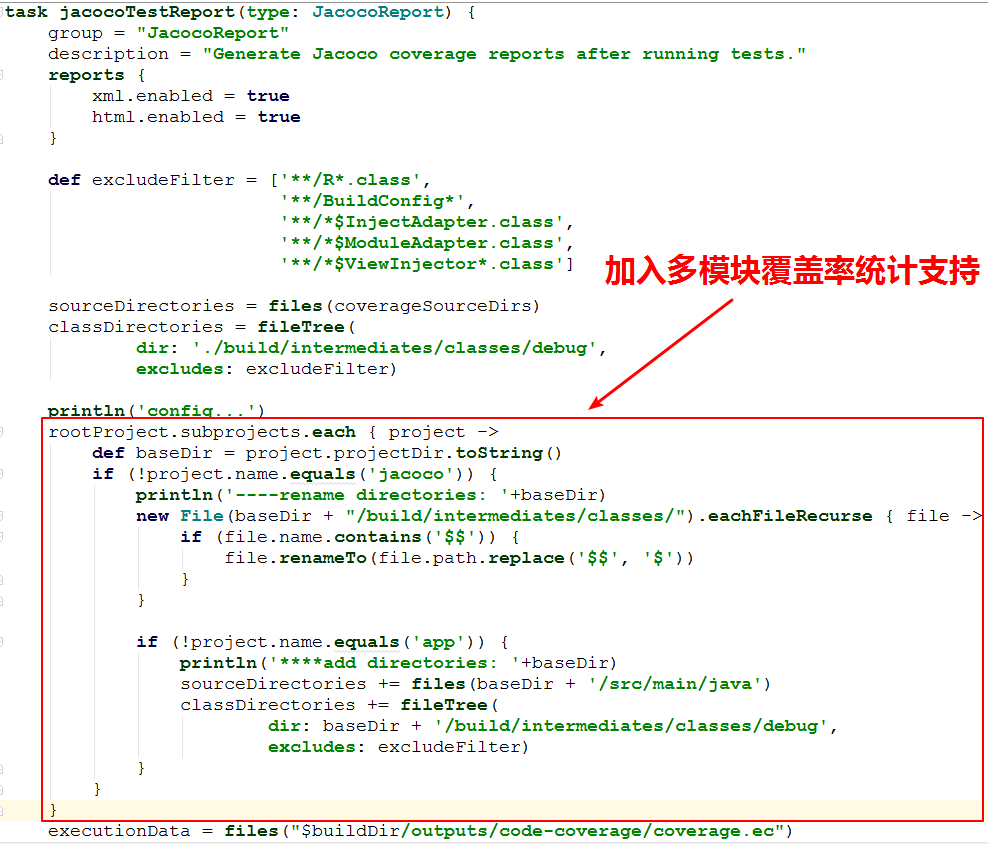


##### 代码覆盖率文件的生成

这里可以自己写一个方法触发覆盖率文件生成，这样的好处是文件生成时机完全由自己控制，特别适合手工测试情形。一般来说都是在APP程序退出时触发，下面是生成覆盖率文件代码片段：



##### 覆盖率生成报告gradle配置



##### 具体使用步骤

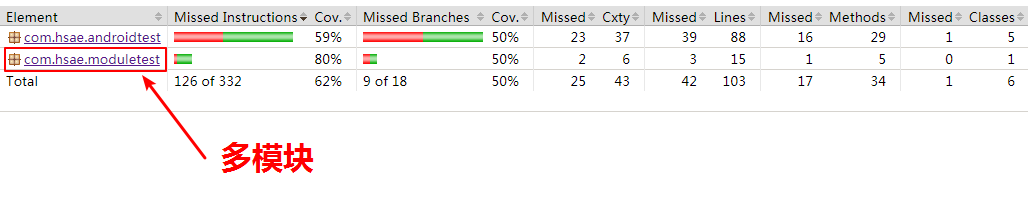
1）安装Debug版程序运行跑一遍，然后退出(会生成覆盖率文件)

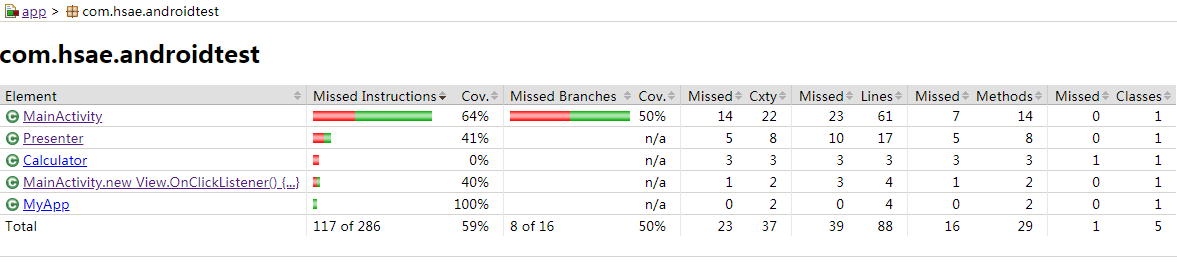
2）将覆盖率文件放到指定位置，运行：gradlew jacocoTestReport

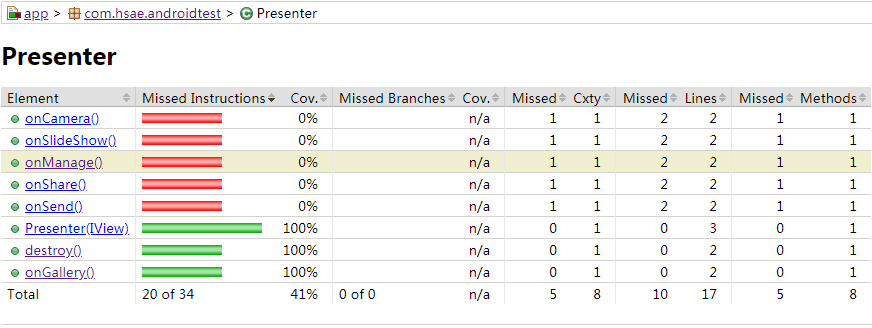
3）到$build/reports/jacoco/jacocoTestReport/html/目录下打开

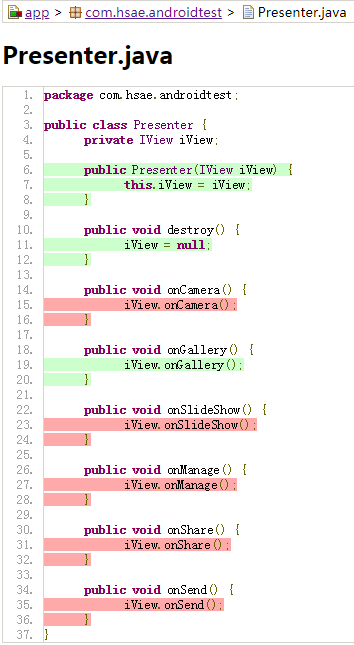
index.html，即可查看到结果

##### 执行结果









#### 关于多模块覆盖率说明

1）每个模块都需要打开jacoco的覆盖率统计开关，不然会失效

2）生成报告的时候需要遍历所有子模块将sourceDirectories

和classDirectories都包含进去

具体请看上面的覆盖率报告gradle配置

#### 更深入使用请参考文档

<https://docs.gradle.org/current/userguide/jacoco_plugin.html>

# 总结

本文简单介绍了Android测试的一些手段，特别是开发人员选择自己进行测试用例编写测试时，需要注意根据不同的测试场景选择合适自己的测试框架，最后介绍了JaCoCo插件来监控测试覆盖程度防止遗漏

# 参考

1. <https://developer.android.google.cn/training/testing/>

2. <https://junit.org/junit4/>

3. <https://github.com/hehonghui/mockito-doc-zh>

4. <https://docs.gradle.org/current/userguide/jacoco_plugin.html>